Towards an Open Set of Real-World Benchmarks for Model Queries and Transformations

Amine Benelallam, Massimo Tisi, Istvan Rath, Benedeck Izso, Dimitrios S. Kolovos

Towards an Open Set of Real-World Benchmarks for Model Queries and Transformations

Amine Benelellam
AtlanMod team
Inria, Mines-Nantes, Lina
Nantes, France
amine.benelallam@inria.fr

Massimo Tisi
AtlanMod team
Inria, Mines-Nantes, Lina
Nantes, France
massimo.tisi@inria.fr

István Ráth
Budapest University of
Technology and Economics
Budapest, Hungary
rath@mit.bme.hu

Benedek Izsó
Budapest University of
Technology and Economics
Budapest, Hungary
izso@mit.bme.hu

Dimitrios S. Kolovos
Enterprise Systems Group
University of York
York, United Kingdom
dimitris.kolovos@york.ac.uk

ABSTRACT

With the growing size and complexity of systems under design, industry needs a generation of Model-Driven Engineering (MDE) tools, especially model query and transformation, with the proven capability to handle large-scale scenarios. While researchers are proposing several technical solutions in this sense, the community lacks a set of shared scalability benchmarks, that would simplify quantitative assessment of advancements and enable cross-evaluation of different proposals. Benchmarks in previous work have been synthesized to stress specific features of model management, lacking both generality and industrial validity. In this paper, we initiate an effort to define a set of shared benchmarks, gathering queries and transformations from real-world MDE case studies. We make these case available to community evaluation via a public MDE benchmark repository.

Categories and Subject Descriptors

D.2.2 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; D.2.8 [**Software Engineering**]: Metrics—*performance measures, Complexity measures*; K.6.3 [**Computing Milieux**]: Software Management—*Software selection*

General Terms

Performance, Experimentation, Measurement

Keywords

Benchmarking, Very Large Models, Model transformations, Model queries

1. INTRODUCTION

Over the past decade, technologies around Model-Driven Engineering (MDE) have been offering a systematic approach for software development on top of models. This elegant approach conquered an interesting popularity and engaged both researchers and industrials. However MDE is not able yet to attract large-scale industrial systems considering the serious scalability issues that the current generation of MDE tools is facing, namely (i) the growing complexity of systems under design, and (ii) the huge amount of data they might represent. Therefore, there is a calling need to develop a new generation of tools capable of managing, querying, and transforming Very Large Models (VLMs). Existing empirical assessment [7, 36, 21] has accredited that necessity indeed.

In spite of the advances on Model Transformation (MT) tools, additional research studies are still needed in order to acquire the attention of large-scale industrial systems. Aiming at improving performance and computation cost, many works have been developed around elaborating new features for existing model transformation engines (e. g., *Incrementality* [18, 33, 9], and *Parallelization* [10, 38, 51]). Others [14] chose to develop new tools based on existing frameworks effectively handling concurrency and parallelization.

One of the most computationally expensive tasks in modeling applications is the evaluation of *model queries*. While there exists a number of benchmarks for queries over relational databases [52] or graph stores [12, 50], modeling tool workloads are significantly different. Specifically, modeling tools use more complex queries than typical transactional systems [29], and the real world performance is affected by response time (i.e. execution time for a specific operation such as validation or transformation) than throughput (i.e. the amount of parallel transactions).

In order to overcome these limitations and achieve scalability in MDE, several works [16, 37] drew research roadmaps advancing the use of the Cloud for distributed and collaborative processing of VLMs. Hence, in order to be able to compare this new generation of tools and measure their performance, it is required to provide a transformation benchmark that takes into consideration the well-known scalability issues in MTs [35]. Likewise, benchmarks might be a good reference to help engineers in choosing what fits the most to their solution while developing new tools.

Most of existing benchmarks [32, 8, 54, 55] were more focused

on other properties than scalability (e. g., Transformation sequence length, Transformation strategy, Matching size etc.). Moreover these benchmarks do not provide neither any clue on how to measure transformation scalability from a theoretical point of view, nor real-world case studies. In contrast to the former benchmarks, Izsó et al. [29] are the first ones to provide a precise metrics to predict the performance of graph queries – based on instance models and query specification –, and therefore results in indicators that help selecting the suitable technology. In addition, these benchmarks are specific to either model transformation or model query. On the other side, there exist some reference benchmarks for databases with a higher level of maturity [15, 49]. In [15], Cattel et al. present a benchmark OO1 to measure the performance of specific characteristics of the most frequently used operations – according to feedbacks from industrials – on engineering objects. This benchmark come with a precise specification, measures, and evaluation indeed. In [49] Schmidt et al. introduce XMark, a benchmark for XML data management that copes with a large range of queries coming from real world case scenarios. Each query comes to stress a particular aspect of XML query engines.

In this paper we present a set of benchmarks for model transformation and query engines. Our proposal is to select a set of transformations/queries from real-world cases and to make them available to large public. Two of the four benchmarks included deal with model transformations, while the other two deal with model queries. Models that feed each one of the benchmarks are of increasing size, also different kinds/extensions. They are either concrete, coming from real projects (i. e., reverse engineered Java project models) or generated using deterministic model instantiators. These instantiators can be easily customized to be able to generate models that suit the benchmarking of a specific feature.

These benchmarks are part of the results of the Scalable Modeling and Model Management on the Cloud (MONDO)[1] research project, that aims at advancing MDE's state of the art to tackle very large models [37]. We plan to involve the community in order to build a larger set of case studies covering additional properties/domains (i. e., verification and simulation of critical systems).

The rest of the paper is organized as follows. In Section 2 we describe benchmarks from the state-of-the-art. In Section 3 we outline the four different sets of benchmarks provided in the scope of the paper. Section 4 describes usage and contribution modalities for the repository. Finally, Section 5 provides a conclusion and recapitulates the future plans for the benchmarks suite.

2. RELATED WORK

A few works in literature [54, 55, 31, 8] proposed benchmarks to assist developers in selecting the most suitable query/transformation technology for their application scenarios. However only one of the existing case studies is explicitly dedicated to the manipulation of very large models ([31]) and none is based on benchmarks queries and transformations based on real-world applications. In this section we give a short overview of the related works, while in the next section we introduce the real-world benchmarks proposed within this paper.

One of the widely used benchmarks in MDE is Grabats'09 Reverse Engineering [31], where Jouault et al. proposed a case study that consists of the definition of program comprehension operators (i. e., queries over source code) using a graph or model transformation tool. One of the main challenges in the definition of program comprehension operators as transformations is scalability with respect to input size. This case study is divided into two independent tasks (i) a simple filtering query that selects a subgraph of its input according to a given condition, and (ii) a complex query that computes a control flow graph and a program dependence graph. These queries are performed over the JDTAST metamodel, the Java metamodel used in early versions of MoDisco [13] to discover Java projects. This benchmark comes with 5 different sets of increasing size ranging from 7×10^5 up to 5×10^9.

The experiment in [54] compares the performance of three model transformation engines: ATL, QVT-R, and QVT-O. This comparison is based on two different transformation examples, targeting meta-models with different structural representations: linear representation (Class2RDBMS) and tree-like representation (RSS2ATOM). The benchmarking set involves randomly generated input models of increasing numbers of elements (up to hundreds of thousands). Like the previous work [55], the benchmark sets are also tuned according to a particular feature such as the size of input models, their complexity (complex interconnection structure) and transformation strategies. In order to study the impact of different implementation strategies in ATL, the Class2RDBMS transformation was implemented in different programming styles. The first one promotes expressing input models navigation in the in the right-hand side of the bindings, the second use ATL attribute helpers, and third uses the imperative part of ATL.

The work [55] is considered one of the early systematic MDE benchmarks dedicated to Graph Transformations (GT). It proposes a methodology for quantitative benchmarking in order to assess the performance of GT tools. The overall aim is to uniformly measure the performance of a system under a deterministic, parametric, and especially reproducible environment. Four tools participated in the experimentation: AGG, PROGRES, FUJABA and DB. Every benchmarking set is tuned according to some features related on one side to the graph transformation paradigms, and on the other side to the surveyed tools. Thus, a benchmarking set is characterized by turning on/off these features. Bergmann et al. extended this benchmark with incrementality. Two kinds of benchmarks kind were carried out, simulation and synchronization, for which, a benchmark-specific generators has been provided. The benchmarks were run over different implementations of pattern matchers, VIATRA/LS (Local Search), VIATRA/RETE, and GEN.NET with the distributed mutual exclusion algorithm.

3. BENCHMARKS SET

The benchmarks set has the purpose to evaluate and validate a proposed query and transformation engine. This set of benchmarks is made public, with the intention to also help both research groups and companies to assess their solutions.

In this section we describe the source, context and properties of the benchmarks. The set of benchmarks is designed to cover the main use cases for queries and transformations in model-driven applications. Table 1 summarizes the characteristics of the four benchmarks in terms of type of computation (query/transformation) and computational complexity (high/low).

Each one of the benchmarks either includes concrete source models, or a model generator that can be used to produce models of different sizes in a deterministic manner. In the latter case, models

[1] http://www.mondo-project.org/

Table 1: Summary of the MONDO WP3 benchmarks

Benchmark	Type	Computational complexity
Train benchmark	query	high
Open-BIM	query/transformation	low
ITM Factory	transformation	high
Transformation zoo	transformation	low

of different sizes can be generated, but seeds are provided to drive the deterministic generators in producing the same models for each user.

Each benchmark in the set is given a reference implementation that has to be considered as a specification of the case semantics. Languages and technologies used for each reference implementation may vary, including MDE-specific and general-purpose technologies.

Finally, while each benchmark defines the source/target relation for each query or transformation, other aspects of the transformation runtime semantics are left open. For instance high-complexity benchmarks can be run in batch or incremental mode, to test different execution properties of the tool under study.

3.1 Train Benchmark

3.1.1 Context and objectives

The Train Benchmark [53, 1] is a macro benchmark that aims to measure batch and incremental query evaluation performance, in a scenario that is specifically modeled after *model validation* in (domain-specific) modeling tools: at first, the entire model is validated, then after each model manipulation (e.g., the deletion of a reference) is followed by an immediate re-validation. The benchmark records execution times for four phases:

1. During the *read* phase, the instance model is loaded from hard drive to memory. This includes the parsing of the input as well as initializing data structures of the tool.
2. In the *check* phase, the instance model is queried to identify invalid elements. This can be as simple as reading the results from cache, or the model can be traversed based on some index. By the end of this phase, erroneous objects need to made available in a list.
3. In the *edit* phase, the model is modified to simulate effects of manual user edits. Here the size of the change set can be adjusted to correspond to small manual edits as well as large model transformations.
4. The re-validation of the model is carried out in the *re-check* phase similarly to the *check* phase.

The Train Benchmark computes two derived results based on the recorded data: (1) *batch validation time* (the sum of the *read* and check phases) represents the time that the user must wait to start to use the tool; (2) *incremental validation time* consists of the *edit* and *re-check* phases performed 100 times, representing the time that the user spent waiting for the tool validation.

3.1.2 Models and metamodels

The Train Benchmark uses a domain-specific model of a railway system that originates from the MOGENTES EU FP7 project, where both the metamodel and the well-formedness rules were defined by railway domain experts. This domain enables the definition of both simple and more complex model queries while it is uncomplicated enough to incorporate solutions from other technological spaces (e.g. ontologies, relational databases and RDF). This allows the comparison of the performance aspects of wider range of query tools from a constraint validation viewpoint.

The instance models are systematically and reproducibly generated based on the metamodel and the defined complex model queries: small instance model fragments are generated based on the queries, and then they are placed, randomized and connected to each other. The methodology takes care of controlling the number of matches of all defined model queries. To break symmetry, the exact number of elements and cardinalities are randomized (with a fixed seed to ensure deterministic reproducibility). In the benchmark measurements, model sizes ranging from a few elements to 13 million elements (objects and references combined) are considered.

This brings artificially generated models *closer to real world instances*, and *prevents query tools from efficiently storing* or caching of instance models. This is important in order to reduce the sampling bias of the experiments. During the generation of the railway system model, errors are injected at random positions. These errors can be found in the check phase of the benchmark, which are reported, and can be corrected during the edit phase. The initial number of constraint violating elements is low (<1% of total elements).

3.1.3 Queries and transformations

Queries are defined informally in plain text (in a tool independent way) and also formalized using the standard OCL language as a reference implementation (available on the benchmark website [1]). The queries range from simple attribute value checks to complex navigation operations consisting of several (4+) joins.

The functionally equivalent variants of these queries are formalized using the query language of different tools applying tool based optimizations. As a result, all query implementations must return (the same set of) invalid instance model elements.

In the *edit* phase, the model is modified to change the result set to be returned by the query in the re-check phase. For simulating manual modifications, the benchmark always performs a hundred random edits (fixed low constant) which increases the number of erroneous elements. An edit operation only modifies one model element at a time - more complex model manipulation is modeled as a series of edits.

The Train Benchmark defines a Java-based framework and application programming interface that enables the integration of additional metamodels, instance models, query implementations and even new benchmark scenarios (that may be different from the original 4-phase concept). The default implementation contains a benchmark suite for queries implemented in Java, Eclipse OCL and EMF-IncQuery.

Measurements are recorded automatically in a machine-processable format (CSV) that is automatically processed by R [2] scripts. An extended version of the Train Benchmark [29] features several (in-

stance model, query-specific and combined) *metrics* that can be used to characterize the "difficulty" of benchmark cases numerically, and – since they can be evaluated automatically for other domain/model/query combinations – allow to compare the benchmark cases with other real-world workloads.

3.2 Open-BIM

3.2.1 Context and objectives

This benchmark includes a model validation and a model transformation in the context of construction models. The construction industry has traditionally communicated building construction information (sites, buildings, floors, spaces, and equipment and their attributes) through drawings with notes and specifications. BIM (Building Information Model), a CAD (Computer Aided Design) method, came to automate that process and enhance its operability according to different tools, actors, etc. within the AECO (Architecture, Engineering, Construction, and Operations) industry. A BIM model is a multi-disciplinary data specific model instance which describes all the information pertinent to a building and its components.

A BIM model is described using the IFC (Industry Foundation Classes) specification, a freely available format to describe, exchange, and share information typically used within the building and facility management industry sector. Intrinsically, IFC is expressed using the EXPRESS data definition language, defined as ISO10303-11 by the ISO TC184/SC4 committee. EXPRESS representations are known to be compact and well suited to include data validation rules within the data specification.

3.2.2 Models and metamodel

The repository contains 8 real-world IFC data files with size ranging from 40MB to 1GB. All the files represent real construction projects and were used in production context. They contain a precise and detailed information about actors, approvals, buildings etc. The data files, in EXPRESS format, are translated into EMF models so that they can be used by EMF-based query and transformation tools.

3.2.3 Queries and transformations

The Open-BIM use case includes a query benchmark and a transformation benchmark:

IFC well-formedness rules. The IFC format describes, using the EXPRESS language, the set of well-formed IFC models. The EXPRESS notation includes, in a single specification, 1) the set of element types and properties allowed in the data file, 2) the set of well-formedness constraints that have to be globally satisfied. When representing an IFC model in an EMF format these two parts of the specification translate to 1) an Ecore metamodel defining element and property types, 2) a set of constraints encoding the well-formedness rules.

This benchmark involves validating the set of well-formedness rules (2) over a given model, model that conforms to the IFC Ecore metamodel (1). An Ecore metamodel is provided, coming from the open-source BIMServer [2] project. The well-formedness rules are given in EXPRESS format and are meant to be translated to the query technology under evaluation.

[2]https://github.com/opensourceBIM/BIMserver

IFC2BIMXML. BIMXML [3] is an XML format describing building data in a simplified spatial building model. The BIMXML XML Schema was developed as an alternative to full scale IFC models to simplify data exchanges between various AEC applications and to connect Building Information Models through Web Services. It is currently used by several primary actors in the CAD construction domain, including Onuma System (Onuma, Inc.), DDS Viewer (Data Design System), vROC, Tokmo, BIM Connect, and various plugins for CAD Applications (Revit, SketchUp, ArchiCAD). The BIMXML specification includes an XML Schema[4] and documents the translation rules from the full IFC specification.

This benchmark involves performing the translation of a full IFC model into the BIMXML format. Ecore metamodels for the source and target models are provided.

3.3 ITM Factory

3.3.1 Context and objectives

This benchmark contains two transformations and a set of queries, each addressing a different phase in a model-driven reverse engineering process. The use case for this benchmark is taken from the Eclipse MoDisco project.

MoDisco (Model Discovery) is the open-source Model Driven Reverse Engineering project lead by the company Soft-Maint. It uses a two steps approach with a **model discovery** followed by **model understanding**. The initial step consists in obtaining a model representation of a specific view on the legacy system, whereas, the second involves extracting useful information from the discovered model. MoDisco requires high efficiency in handling large models, especially these involved in reverse engineering of legacy systems.

3.3.2 Models and metamodel

Thanks to the MoDisco Java discoverer, we are able to extract Java models up to 1.3GB, that conform to the Java metamodel [39] defined in MoDisco (refinement of the JDTAST metamodel). Those models are the input of the Java2KDM and Java code quality transformations, while, KDM output models are inputs for the KDM2UML transformation. It is also possible to retrieve directly KDM models using MoDisco. Because of confidentiality agreements, Soft-Maint is not able to publish instance models derived from their commercial projects. For this reason we choose to derive instance models from the source code of open-source projects, specifically from the Eclipse JDT plugins (org.eclipse.jdt.*). This does not affect the relevance of the benchmark, as these plugins are written by experienced developers with a quality standard that is comparable to commercial projects.

Table 2 depicts the different models recovered against the discovered plugins.

3.3.3 Queries and transformations

Java2KDM. This transformation takes place at beginning of almost every modernization process of a Java legacy system, it comes just after the discovery of the Java model from Java projects (plugins) using the MoDisco Java Discoverer. This transformation generates a KDM [44] (Knowledge Discovery Metamodel) model that defines common metadata required for deep semantic integration of

[3]http://bimxml.org/

[4]http://www.bimxml.org/xsd/001/bimxml-001.xsd

Set1	org.eclipse.jdt.apt.pluggable.core
Set2	Set1 + org.eclipse.jdt.apt.pluggable.core
Set3	Set2 + org.eclipse.jdt.core+ org.eclipse.jdt.compiler + org.eclipse.jdt.apt.core
Set4	Set3 + org.eclipse.jdt.core.manipulation + org.eclipse.jdt.launching + org.eclipse.jdt.ui + org.eclipse.jdt.debug
Set5	org.eclipse.jdt.* (all jdt plugins)

Table 2: Discovered plugins per set

application life-cycle management tools. Java2KDM transformation is useful when the only available information on a Java source code is contained in a Java model, even without the source code it is then possible to get a KDM representation. This intermediate model provides useful and precise information that can be used to produce additional types of models.

KDM2UML. Based on the previously generated model, this transformation generates a UML diagram in order to allow integrating KDM-compliant tools (i. e., discoverers) with UML-compliant tools (e.g. modelers, model transformation tools, code generators, etc.).

Java code quality. This set of code quality transformations identify well-known anti-patterns in Java source code and fix the corresponding issues by a model transformation. The input format of the transformations is a model conforming to the Java metamodel. For a specification for the transformations we refer the reader to the implementations of these fixes in well-known code-analysis tools like CheckStyle and PMD. Table 3 summarizes the references to the documentation for each code fix considered in this benchmark.

3.4 ATL Transformation Zoo

3.4.1 Context and objectives

The ATL project maintains a repository of ATL transformations produced in industrial and academic contexts (ATL Transformation Zoo [28]). These transformations are representative of the use of model transformations for low-complexity tasks (i.e., low number of transformation rules, lack of recursion, etc. . .).

In this benchmark we select a subset of the transformations in the ATL Transformation Zoo based on their quality level and usage in real-world applications. We specifically include only transformations that may be used in production environments. We automatize the sequential execution of this subset and the generation of performance analysis data.

3.4.2 Models and metamodels

For the aforementioned transformations, we do not have large enough models that conform to the respective metamodels, and as such we make use of a probabilistic model instantiator. This instantiator takes as parameter a generation configuration specified by the user. A generation configuration holds information such as 1) metaclasses that should (not) be involved in the generation, 2) probability distributions to establish how many instances should be generated for each metaclass, and which values should be assigned to structural features. We provide a default generation configuration, using uniform probability distributions for each meta-class and structural feature. For some transformations we provide ad-hoc probability distributions, exploiting domain knowledge over the instances of the corresponding metamodel.

A generation configuration may come also with a seed that makes the generation deterministic and reproducible. For each one of the built-in generation configurations we provide a seed, producing the exact set of models we used during our experimentation.

3.4.3 Queries and transformations

Ant to Maven. Ant [4] is an open source build tool (a tool dedicated to the assembly of the different pieces of a program) from the Apache Software Foundation. Ant is the most commonly used build tool for Java programs. Maven [5] is another build tool created by the Apache Software Foundation. It is an extension of Ant because ant Tasks can be used in Maven. The difference from Ant is that a project can be reusable. This transformation [22] generates a file for the build tool Maven starting from a file corresponding to the build tool Ant.

CPL2SPL. CPL (Call Processing Language) is a standard scripting language for the SIP (Session Initiation Protocol) protocol. It offers a limited set of language constructs. CPL is supposed to be simple enough so that it is safe to execute untrusted scripts on public servers [30]. SPL programs are used to control telephony agents (e.g. clients, proxies) implementing the SIP (Session Initiation Protocol) protocol. Whereas, the CPL has an XML-based syntax, the *CPL2SPL* transformation [24], provides an implementation of CPL semantics by translating CPL concepts into their SPL equivalent concepts.

Graphcet2PetriNet. This transformation[25] establishes a bridge between grafcet [17], and petri nets [43]. It provides an overview of the whole transformation sequence that enables to produce an XML petri net representation from a textual definition of a grafcet in a PNML format, and the other way around.

IEEE1471 to MoDAF. This transformation example [3] realizes the transformation between IEEE1471-2000 [34] and MoDAF-AV [11]. The IEEE1471 committee prescribes a recommended practice for the design and the analysis of Architecture of Software Intensive Systems. It fixes a terminology for System, Architecture, Architectural Description, Stakeholder, Concerns, View ,and Viewpoints concepts. MoDAF (Ministry of Defense Architecture Framework) is based on the DoDAF (Department of Defense Architecture Framework). DoDAF is a framework to design C4ISR systems. MoDAF-AV (Architecture View) used several concepts defined in the IEEE1471.

Make2Ant. Make (the most common build tool) is based on a particular shell or command interface and is therefore limited to the type of operating systems that use that shell. Ant uses Java classes rather than shell-based commands. Developers use XML to describe the modules in their program build. This benchmark [26] describes a transformation from a Makefile to an Ant file.

MOF2UML. The MOF (Meta Object Facility)[45] is an OMG standard enabling the definition of metamodels through common semantics. The UML (Unified Modeling Language) Core standard is the OMG common modeling language. Although, MOF is primarily designed for metamodel definitions and UML Core for the design of models, the two standards define very close notions. This example [27] describes a transformation enabling to pass from the MOF to the UML semantics. The transformation is based on the UML Profile for MOF OMG specification.

OCL2R2ML. The OCL to R2ML transformation scenario [41] describes a transformation from OCL (Object Constraint Language) [46] metamodel (with EMOF metamodel) into a R2ML (REWERSE I1 Rule Markup Language) metamodel. The Object Constraint Language (OCL) is a language that enables one to describe expressions and constraints on object-oriented (UML and MOF) models and other object modeling artifacts. An expression is an indication or specification of a value. A constraint is a restriction on one or more values of (part of) an object-oriented model or system. REWERSE I1 Rule Markup Language (R2ML) is a general web rule markup language, which can represent different rule types: integrity, reaction, derivation and production. It is used as pivotal metamodel to enable sharing rules between different rule languages, in this case with the OCL.

UML2OWL. This scenario [42] presents an implementation of the OMG's ODM specification. This transformation is used to produce an OWL ontology, and its *OWL Individuals* from an UML Model, and its *UML Instances*.

BibTeXML to DocBook. The *BibTeXML to DocBook* example [23] describes a transformation of a BibTeXML [47] model to a DocBook [56] composed document. BibTeXML is an XML-based format for the BibTeX bibliographic tool. DocBook, as for it, is an XML-based format for document composition.

DSL to EMF. This example [6] provides a complete overview of a transformation chain example between two technical spaces: Microsoft DSL Tools [40] and EMF. The aim of this example is to demonstrate the possibility to exchange models defined under different technologies. In particular, the described bridges demonstrate that it should be possible to define metamodels and models using both Microsoft DSL Tools and Eclipse EMF technologies. The bridge between MS/DSL and EMF spans two levels: the metamodel and model levels. At the level of metamodels, it allows to transform MS/DSL domain models to EMF metamodels. At the level of models, the bridge allows transforming MS/DSL models conforming to domain models to EMF models conforming to EMF metamodels. At both levels, the bridge operates in both directions. A chain of ATL-based transformations is used to implement the bridge at these two levels. The benefit of using such a bridge is the ability to transpose MS/DSL work in EMF platform, and inversely.

4. THE MDE BENCHMARK REPOSITORY

The *MDE Benchmark repository* is the central storage area where the artifacts of the benchmarks are archived for public access. These artifacts, mainly text files, comprise large models and metamodels – typically represented in their XMI serialization – and model transformations. To increase the visibility of these files we have chosen to make them publicly available through the OpenSourceProjects.eu (OSP) platform. The OSP platform is a software forge dedicated to hosting Open Source projects created within EU research projects.

The OSP platform provides, among other tools, a Git revision control system (RCS). Git repositories hosted in the OSP platform can be easily navigated by a Web interface.

4.1 Benchmark structure

The *MDE Benchmark repository* is located at [48]. Inside this repository every top level resource corresponds to a git submodule, each, representing a different case study held in a separate git repository.

Related resources for benchmarking a specific feature of a transformation engine are grouped in *projects*. A *project* is a self-contained entity, and can be considered as the basic benchmarking unit. *Projects* share a common internal structure that includes a short case description and a set of (optional) folders:

Short case description — A mandatory human-readable file describes the details of the test case, the file and directory structure, and any other important information (e. g., test cases can evolve and additional information not considered at the point of writing this document may be needed for executing the benchmark).

Documentation — This directory stores the documentation about the test case. The documentation of a test case may include, among other information, a detailed description of the test case, the foundations of the feature under testing, the building and execution instructions, etc.

Queries and Transformations — This directory stores the queries and transformations, in source code form, that stress the feature under testing.

Models — This directory contains the model and metamodel descriptions involved in the test transformation(s).

Input data — This directory contains the input data to be used by the test case(s).

Expected data — In this directory we store the files that contain the expected values that must be returned by the transformation. The expected data are compared with the actual output of the transformation to determine if the test execution has been successful or not.

Source code — In some situations, test cases may require additional code (such as Java code) to be executed. For example, test cases may be automatically launched with the help of third party libraries (such as JUnit), or test cases may execute external code following a black-box scheme. In this situations the additional code should be placed inside the `/src` directory.

Libraries — This directory is used to store any additional third party library (usually a binary file) required by the test case.

Scripts — Build and execution scripts should be placed under the `/build` directory. Examples of such scripts are Ant files [4], Maven files [5], Makefiles [20], bash shell scripts [19].

4.2 Submission guidelines

In order to increase the quality and soundness of the test cases available in the *MDE Benchmark repository*, we plan to keep it open to further submissions from the MDE community.

We have defined a simple set of guidelines that must be followed when contributing a new case study to guarantee that the quality of the repository is maintained. Specifically:

- New contributions must include a comprehensive description of the case study. A rationale for its inclusion must be provided, specially focusing on the differential aspects of the proposed case study, compared to the already included benchmarks.
- The sources, models, documentation and utility scripts must be organized as described in Section 4.1.
- Contributions must be sent to the address `mondo_team@opengroup.org` for their evaluation and approval.

5. CONCLUSION

This paper introduces the first open-set benchmark gathered from real-world cases to stress scalability issues in model transformation and query engines. These benchmark suite comes not only with the aim of providing a point of reference against which industrials and researchers might compare between different technologies to choose what could suit their needs, but also to motivate the MDE community to be part of its extension and contribute with additional cases not covered by this set.

In our future work we plan to furnish a feature-based organization of the benchmark in order to ease its use and enable efficient profit. We also intend to provide theoretical background on how to measure transformations scalability. Another point would be to optimize model instances generation to allow the generation of bigger models, also to contribute to the repository with a live/real-time instantiators for the consideration of infinite model transformations.

6. ACKNOWLEDGMENTS

This work is partially supported by the MONDO (EU ICT-611125) project. The authors would like to thank UNINOVA and Soft-Maint for their inputs, materials, and valuable discussions.

7. REFERENCES

[1] The train benchmark website. `https://incquery.net/publications/trainbenchmark/full-results`, 2013.

[2] The R project for statistical computing. `http://www.r-project.org/`, 2014.

[3] Albin Jossic. ATL Transformation Example: IEEE1471 to MoDAF, 2005. URL: `http://www.eclipse.org/atl/atlTransformations/IEEE1471_2_MoDAF/IEEE1471_2_MoDAF.doc`.

[4] Apache. Apache ant, 2014. URL: `http://ant.apache.org/`.

[5] Apache. Apache maven project, 2014. URL: `http://maven.apache.org/`.

[6] ATLAS group – LINA & INRIA. The Microsoft DSL to EMF ATL transformation , 2005. URL: `http://www.eclipse.org/atl/atlTransformations/DSL2EMF/ExampleDSL2EMF%5Bv00.01%5D.pdf`.

[7] P. Baker, S. Loh, and F. Weil. Model-driven engineering in a large industrial contextŮmotorola case study. In *Model Driven Engineering Languages and Systems*, pages 476–491. Springer, 2005.

[8] G. Bergmann, Á. Horváth, I. Ráth, and D. Varró. A benchmark evaluation of incremental pattern matching in graph transformation. In *Graph Transformations*, pages 396–410. Springer, 2008.

[9] G. Bergmann, D. Horváth, and Á. Horváth. Applying incremental graph transformation to existing models in relational databases. In *Graph Transformations*, pages 371–385. Springer, 2012.

[10] G. Bergmann, I. Ráth, and D. Varró. Parallelization of graph transformation based on incremental pattern matching. *Electronic Communications of the EASST*, 18, 2009.

[11] B. Biggs. Ministry of defence architectural framework (modaf). 2005.

[12] C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web & Information Systems*, 5(2):1–24, 2009.

[13] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012 – 1032, 2014.

[14] L. Burgueño, J. Troya, M. Wimmer, and A. Vallecillo. On the concurrent execution of model transformations with linda. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, page 3. ACM, 2013.

[15] R. G. G. Cattell and J. Skeen. Object operations benchmark. *ACM Trans. Database Syst.*, 17(1):1–31, Mar. 1992.

[16] C. Clasen, M. D. Del Fabro, and M. Tisi. Transforming very large models in the cloud: a research roadmap. In *First International Workshop on Model-Driven Engineering on and for the Cloud*, 2012.

[17] R. David. Grafcet: A powerful tool for specification of logic controllers. *Control Systems Technology, IEEE Transactions on*, 3(3):253–268, 1995.

[18] H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software & Systems Modeling*, 8(1):21–43, 2009.

[19] GNU. Bourne-Again SHell manual, 2014. URL: `http://www.gnu.org/software/bash/manual/`.

[20] GNU. GNU 'make', 2014. URL: `http://www.gnu.org/software/make/manual/`.

[21] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 471–480. ACM, 2011.

[22] INRIA. ATL Transformation Example: Ant to Maven, 2005. URL: `http://www.eclipse.org/atl/atlTransformations/Ant2Maven/ExampleAnt2Maven%5Bv00.01%5D.pdf`.

[23] INRIA. ATL Transformation Example: BibTeXML to DocBook, 2005. URL: `http://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook%5Bv00.01%5D.pdf`.

[24] INRIA. ATL Transformation Example: CPL to SPL, 2005. URL: `http://www.eclipse.org/atl/atlTransformations/CPL2SPL/README.txt`.

[25] INRIA. ATL Transformation Example: Grafcet to Petri Net, 2005. URL: `http://www.eclipse.org/atl/atlTransformations/Grafcet2PetriNet/`

ExampleGrafcet2PetriNet[v00.01].pdf.

[26] INRIA. ATL Transformation Example: Make to Ant, 2005. URL: http://www.eclipse.org/atl/atlTransformations/Make2Ant/ExampleMake2Ant[v00.01].pdf.

[27] INRIA. ATL Transformation Example: MOF to UML, 2005. URL: http://www.eclipse.org/atl/atlTransformations/MOF2UML/ExampleMOF2UML[v00.01].pdf.

[28] Inria. Atl transformation zoo, 2014. URL: http://www.eclipse.org/atl/atlTransformations/.

[29] B. Izsó, Z. Szatmári, G. Bergmann, Á. Horváth, and I. Ráth. Towards precise metrics for predicting graph query performance. In *2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 412–431, Silicon Valley, CA, USA, 11/2013 2013. IEEE.

[30] F. Jouault, J. Bézivin, C. Consel, I. Kurtev, F. Latry, et al. Building dsls with amma/atl, a case study on spl and cpl telephony languages. In *ECOOP Workshop on Domain-Specific Program Development*, 2006.

[31] F. Jouault, J. Sottet, et al. An amma/atl solution for the grabats 2009 reverse engineering case study. In *5th International Workshop on Graph-Based Tools, Zurich, Switzerland*, 2009.

[32] F. Jouault, J.-S. Sottet, et al. An amma/atl solution for the grabats 2009 reverse engineering case study. In *5th International Workshop on Graph-Based Tools, Grabats*, 2009.

[33] F. Jouault and M. Tisi. Towards incremental execution of atl transformations. In *Theory and Practice of Model Transformations*, pages 123–137. Springer, 2010.

[34] E. Jouenne and V. Normand. Tailoring ieee 1471 for mde support. In *UML Modeling Languages and Applications*, pages 163–174. Springer, 2005.

[35] D. S. Kolovos, R. F. Paige, and F. A. Polack. The epsilon transformation language. In *Theory and practice of model transformations*, pages 46–60. Springer, 2008.

[36] D. S. Kolovos, R. F. Paige, and F. A. Polack. The grand challenge of scalability for model driven engineering. In *Models in Software Engineering*, pages 48–53. Springer, 2009.

[37] D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, D. Varró, M. Tisi, et al. A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, page 2. ACM, 2013.

[38] G. Mezei, T. Levendovszky, T. Mészáros, and I. Madari. Towards truly parallel model transformations: A distributed pattern matching approach. In *EUROCON 2009, EUROCON'09. IEEE*, pages 403–410. IEEE, 2009.

[39] MIA-Software. Modiscojava metamodel (knowledge discovery metamodel) version 1.3, 2012. URL: http://dev.eclipse.org/svnroot/modeling/org.eclipse.mdt.modisco/main/branches/0_11/org.eclipse.gmt.modisco.java/model/java.ecore.

[40] Microsoft Corp. The DSL tools, 2014. URL: http://msdn.microsoft.com/vstudio/DSLTools/.

[41] Milan Milanovic. ATL Transformation Example: OCL to R2ML, 2005. URL: http://www.eclipse.org/atl/atlTransformations/OCL2R2ML/README.txt.

[42] Milan Milanovic. ATL Transformation Example: UML to OWL, 2005. URL: http://www.eclipse.org/atl/atlTransformations/UML2OWL/README.txt.

[43] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[44] OMG (Object Management Group). Kdm (knowledge discovery metamodel) version 1.3, 2011. URL: http://www.omg.org/spec/KDM/1.3/.

[45] OMG (Object Management Group). Mof (meta object facility) version 2.4, 2011. URL: http://www.omg.org/spec/MOF/2.4.

[46] OMG (Object Management Group). Ocl (object constraint language) v2.0, 2011. URL: http://www.omg.org/spec/OCL/2.0/PDF.

[47] L. Previtali, B. Lurati, and E. Wilde. Bibtexml: An xml representation of bibtex. In V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, editors, *WWW Posters*, 2001.

[48] M. Project. Transformation benchmarks, 2014. URL: http://opensourceprojects.eu/p/mondo/d31-transformation-benchmarks/.

[49] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 974–985. VLDB Endowment, 2002.

[50] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL performance benchmark. In *Proc. of the 25th International Conference on Data Engineering*, pages 222–233, Shanghai, China, 2009. IEEE.

[51] M. Tisi, S. Martinez, and H. Choura. Parallel execution of atl transformation rules. In *Model-Driven Engineering Languages and Systems*, pages 656–672. Springer, 2013.

[52] Transaction Processing Performance Council (TPC). TPC-C Benchmark, 2010.

[53] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming*, 2014.

[54] M. Van Amstel, S. Bosems, I. Kurtev, and L. F. Pires. Performance in model transformations: experiments with atl and qvt. In *Theory and Practice of Model Transformations*, pages 198–212. Springer, 2011.

[55] G. Varro, A. Schurr, and D. Varro. Benchmarking for graph transformation. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 79–88. IEEE, 2005.

[56] N. Walsh and R. Hamilton. *DocBook 5: The Definitive Guide*. Definitive Guide Series. O'Reilly Media, 2010.

Table 3: List of Java code quality fixes

Rule	Documentation
ConstantName	http://checkstyle.sourceforge.net/config_naming.html#ConstantName
LocalFinalVariableName	http://checkstyle.sourceforge.net/config_naming.html#LocalFinalVariableName
LocalVariableName	http://checkstyle.sourceforge.net/config_naming.html#LocalVariableName
MemberName	http://checkstyle.sourceforge.net/config_naming.html#MemberName
MethodName	http://checkstyle.sourceforge.net/config_naming.html#MethodName
PackageName	http://checkstyle.sourceforge.net/config_naming.html#PackageName
ParameterName	http://checkstyle.sourceforge.net/config_naming.html#ParameterName
StaticVariableName	http://checkstyle.sourceforge.net/config_naming.html#StaticVariableName
TypeName	http://checkstyle.sourceforge.net/config_naming.html#TypeName
AvoidStarImport	http://checkstyle.sourceforge.net/config_imports.html#AvoidStarImport
UnusedImports	http://checkstyle.sourceforge.net/config_imports.html#UnusedImports
RedundantImport	http://checkstyle.sourceforge.net/config_imports.html#RedundantImport
ParameterNumber	http://checkstyle.sourceforge.net/config_sizes.html#ParameterNumber
ModifierOrder	http://checkstyle.sourceforge.net/config_modifier.html#ModifierOrder
RedundantModifier	http://checkstyle.sourceforge.net/config_modifier.html#RedundantModifier
AvoidInlineConditionals	http://checkstyle.sourceforge.net/config_coding.html#AvoidInlineConditionals
EqualsHashCode	http://checkstyle.sourceforge.net/config_coding.html#EqualsHashCode
HiddenField	http://checkstyle.sourceforge.net/config_coding.html#HiddenField
MissingSwitchDefault	http://checkstyle.sourceforge.net/config_coding.html#MissingSwitchDefault
RedundantThrows	http://checkstyle.sourceforge.net/config_coding.html#RedundantThrows
SimplifyBooleanExpression	http://checkstyle.sourceforge.net/config_coding.html#SimplifyBooleanExpression
SimplifyBooleanReturn	http://checkstyle.sourceforge.net/config_coding.html#SimplifyBooleanReturn
FinalClass	http://checkstyle.sourceforge.net/config_design.html#FinalClass
InterfaceIsType	http://checkstyle.sourceforge.net/config_design.html#InterfaceIsType
VisibilityModifier	http://checkstyle.sourceforge.net/config_design.html#VisibilityModifier
FinalParameters	http://checkstyle.sourceforge.net/config_misc.html#FinalParameters
LooseCoupling	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/typeresolution.html#LooseCoupling
SignatureDeclareThrowsException	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/typeresolution.html#SignatureDeclareThrowsException
DefaultLabelNotLastInSwitchStmt	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#DefaultLabelNotLastInSwitchStmt
EqualsNull	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#EqualsNull
CompareObjectsWithEquals	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#CompareObjectsWithEquals
PositionLiteralsFirstInComparisons	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/design.html#PositionLiteralsFirstInComparisons
UseEqualsToCompareStrings	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/strings.html#UseEqualsToCompareStrings
IntegerInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#IntegerInstantiation
ByteInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#ByteInstantiation
ShortInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#ShortInstantiation
LongInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#LongInstantiation
BooleanInstantiation	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/migrating.html#BooleanInstantiation
SimplifyStartsWith	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/optimizations.html#SimplifyStartsWith
UnnecessaryReturn	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/unnecessary.html#UnnecessaryReturn
UnconditionalIfStatement	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/basic.html#UnconditionalIfStatement
UnnecessaryFinalModifier	http://pmd.sourceforge.net/pmd-5.1.0/rules/java/unnecessary.html#UnnecessaryFinalModifier